PATENT APPLICATION

COMPUTER SYSTEM SECURITY VIA DYNAMIC ENCRYPTION

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is a continuation-in-part of U.S. Patent Application Serial No. 10/387,711, entitled "Computer System Security via Dynamic Encryption," filed on March 13, 2003, and claims the benefit of the filing date thereof. The entire specification of the parent application is incorporated herein by reference.

BACKGROUND OF THE INVENTION

Field of the Invention (Technical Field):

The present invention relates to the field of computer system security, more particularly to a dynamic data encryption and node authentication method and system that distributes the complexity of the encryption algorithm over the dynamics of data exchange and involves full synchronization of encryption key regeneration at system nodes, independent of the node clocks.

Background Art:

The fundamental objective of cryptography is to enable users to communicate securely via an insecure shared data communication channel or system environment, maintaining data integrity, privacy, and user authentication. Over the past century, various cryptography systems have been developed which require a great deal of time to break even with large computational power. However, if an intruder obtains the encryption key, the encryption mechanism, and probably the entire system security, is compromised and a new key is required.

In order to make an encryption system nearly impenetrable to an intruder, two strategies are commonly used: 1) a long encryption key, and/or 2) a complex encryption function. A key of length n bits has a 2^n search space. Therefore, for large values of n an intruder needs to spend more than a lifetime to break the cipher. Also, simpler encryption functions provide a less secure encryption system. For instance, an encryption code that applies the logic XOR function is easy to decipher no matter how long the key length is. This is because the XOR operation is performed on one bit of data and its corresponding bit from the encryption key, one bit at a time. The deciphering approach of such simple encryption functions by an intruder is based on the divide-and-conquer mechanism. The intruder first deciphers individual key fragments, which is relatively uncomplicated to accomplish due to the simple linearity of the XOR function, then reconstructs the entire key once all of the individual fragments are obtained. It is more difficult to apply such a divide-and-conquer approach to break the key of a nonlinear exponential encryption function, such as used in the Rivest-Shamir-Adelman (RSA) system.

At present, there are two major cryptography system philosophies: 1) symmetric systems (static or semi-dynamic key), and 2) public key systems (static key). In symmetric systems, e.g., DES, AES, etc., a key is exchanged between the users, the sender and receiver, and is used to encrypt and decrypt the data. There are three major problems with symmetric systems. First, exchanging the key between users introduces a security loophole. In order to alleviate such a problem, the exchanged key is encrypted via a secure public key cryptography system. Second, the use of only one static encryption key makes it easier for an intruder to have an ample amount of time to break the key. This issue is addressed by the use of multiple session keys that are exchanged periodically. Third, and more importantly is the susceptibility to an "insider" attack on the key. This is referred to as the "super user" spying on the "setting duck" static key inside the system, where the time window

between exchanging keys might be long enough for a super user, who has a super user privilege, to break in and steal the key.

In the RSA public key cryptography system, a user (U) generates two related keys, one is revealed to the public, deemed the "public" key, to be used to encrypt any data to be sent to U. The second key is private to U, called the "private" key, and is used to decrypt any data received at U, which was encrypted with the corresponding public key. The RSA cryptography system generates large random primes and multiplies them to get the public key. It also uses a complex encryption function such as mod and exponential operations. As a result, this technique is unbreakable in the lifetime of a human being for large keys, e.g., higher than 256 bits, and also eliminates the problem of the insecure exchange of symmetric keys, as in a DES system. However, the huge computational time required by RSA encryption and decryption, in addition to the time required to generate the keys, is not appealing to the Internet user community. Thus, RSA cryptography is mainly used as "one shot" solid protection of the symmetric cryptography key exchange.

In the RSA public key system, if a first user (U_A) requests a secure communication with a second user (U_B), the latter will generate a pair of encryption keys: public E_B and private D_B. An internal super user spy (S), with a helper (H) intruding on the communication line externally, can easily generate its own pair of keys, a public E_S and private D_S, and pass D_S and E_B to H. Then S can replace the public key E_B with its own public key E_S. Thus, all data moving from U_A to U_B will be encrypted using E_S instead of E_B. Now H can decrypt the cipher text moving between U_A and U_B using the private key D_S, store it, and re-encrypt it using the original E_B, in order for U_B to receive and decrypt it without any knowledge of the break that occurred in the middle. Such an attack is typically called the "super-user-in-the-middle" attack.

Even though they are secure against outsider attack, both the symmetric and public key cryptography systems are still vulnerable to insider attacks. By obtaining the key at any time of a secure session, an intruder can decipher the entire exchanged data set, past and future. Further, a super user can easily steal a static symmetric key and send it to an outside intruder to sniff and decrypt the cipher text, particularly in the DES and AES systems.

A common way to protect a static encryption key is to save it under a file with restricted access. This restriction is not enough, however, to prevent a person with super-user privilege from accessing the static key in the host file. Even when keys are changed for each communication session, for example in the Diffie-Hufman system, there is a time window enough for the super-user to obtain the semi-static key. In most crypto systems, once the key is found the previous and future communicated data are no longer secure.

Various other attempts have been made to circumvent intrusion by outside users through encryption of communicated data. Examples of such methods include that described in U.S. Patent No. 6,105,133 to Fielder, et al., entitled, "Bilateral Authentication and Encryption System;" U.S. Patent No. 6,049,612 also to Fielder, et al., entitled, "File Encryption Method and System;" and U.S. Patent No. 6,070,198 to Krause, et al., entitled, "Encryption with a Streams-Based Protocol Stack." While the techniques described in these patents may be useful in preventing unwanted intrusion by outsiders, they are still prone to attack by the super-user-in-the-middle.

The present invention alleviates the problems encountered in the prior art, providing continuous encryption key modification. A new key is generated from a previous key as well as from a previously encrypted data record. The regenerated key is then used to encrypt the subsequent data record. The key lifetime is equal to the time span of record encryption, reducing the possibility that an intruder will break the key or that a super-user will copy the key. The present invention also alleviates

the "super-user-in-the-middle" attack. An intruder must obtain an entire set of keys, at the right moment, without being noticed, in order to decrypt the entire ciphered message.

The present invention also reduces computational overhead by breaking the complexity of the encryption function and shifting it over the dynamics of data exchange. Speed is also improved through the use of a simple XOR logic encryption function. A shuffling mechanism based on a dynamic permutation table, which is generated from the current session key, is coupled with the XOR logic operation, to strengthen the encryption function. Encryption is fully automated and all parties, the source user, destination user, and central authority, are clock-free synchronized, and securely authenticated, at all times. The dynamic key encryption system of the present invention is deployable at any level of a system, as there is complete synchronization between parties.

The present invention further eliminates the possibility of an intruder obtaining additional keys, and therefore additional data, in the circumstance where an intruder is able to break one of the keys.

A previously encrypted data record is combined with a previous key to create a new key for encrypting a subsequent data record. Hence, if an intruder were to break one key, the intruder could only decrypt its corresponding data record and nothing more, past or future ciphered data.

SUMMARY OF THE INVENTION (DISCLOSURE OF THE INVENTION)

The present invention is a method of providing a secure data stream between system nodes. The method includes encrypting data at a node with an encryption key, or dynamic session key (*DSK*), selecting encrypted data, and regenerating a new *DSK* with an encryption key and selected encrypted data. Selecting encrypted

data includes selecting encrypted data using a byte from a previous encryption key as a seed of random generation.

The new *DSK* is generated by performing a logic operation on a previous *DSK* and selected previously encrypted data record. Preferably, the logic operation is an XOR operation. The previously encrypted data record and previous *DSK* are XORed to form an expanded key, *ExpK*. Bytes are randomly selected from the *ExpK* to generate the new *DSK*, using a byte from a previous *DSK* as a seed of random generation.

A data record is encrypted with a *DSK* by performing a logic XOR operation on the data and *DSK* to form a temporary cipher. Portions of the cipher are then permuted to form another cipher, which is then transmitted over a data stream to the destination user node.

Blocks of *n* previously encrypted data records of a time increment (T), each record of *m* bytes, and a corresponding number of *n DSK*s, each of *m* bytes, are combined to form *n* new *DSK*s. The *n* new *DSK*s are then used to encrypt the subsequent block of *n* data records. The process continues until all data records are encrypted and transmitted from the source user node.

The method further comprises the step of receiving encrypted data at a destination user node and decrypting the received encrypted data with a *DSK*. Once the encrypted data is decrypted, new *DSK*s are regenerated at the destination user node using selected decrypted data and a previous *DSK*. A central authority node is used to assure that both the source and destination nodes begin with the same initial *DSK* so that the destination node can properly decrypt the received encrypted data.

The present invention is further a system for providing a secure data stream between a source programmable apparatus and a destination programmable apparatus. The system comprises: a source programmable apparatus; a data stream

created by the source programmable apparatus; means for encrypting data of the data stream with a *DSK*; and means for regenerating a new *DSK* using selected previously encrypted data. The system also includes a destination programmable apparatus in electrical communication with the source programmable apparatus; means for transmitting encrypted data to the destination programmable apparatus; means for decrypting the encrypted data received at the destination programmable apparatus with a *DSK*; and means for regenerating a new *DSK* using selected previously decrypted data.

A primary object of the present invention is to provide a dynamic encryption method and system having no static keys, public or private, that are susceptible to a security breach. Another primary object of the invention is to provide improved security to a data stream between a source and destination user, in particular to provide improved security against a super-user having insider privileges. Another primary object of the present invention is to provide such improved security at a high speed, in a secure system environment, via mutually authenticated users and *CA*s. Yet another primary object of the present invention is to provide a dynamic encryption method and system wherein the breaking of one key by an intruder will aid only in decrypting its corresponding data record and nothing more, past or future ciphered data.

A primary advantage of the present invention is that it is fully automated, with all system nodes synchronized and mutually authenticated, to ensure security. Another primary advantage of the invention is that it is simple and fast, yet secure against spying by an internal super-user or outside intruder due to the large number of dynamic keys (*n*) that would need to be broken or compromised -- one key per ciphered data record, (*n*) parallel sessions of encryption, and zero entropy of the ciphered text. Yet another primary advantage of the invention is that it minimizes the exchange of keys between users and/or the *CA*. Still yet another advantage of the

invention is that an initial *DAK* is securely exchanged between a user and *CA* which is continuously regenerated during the entire life of the user, allowing the user and *CA* to synchronize (realign *DAK*s) and authenticate to one another as needed for a communication session or when there is a disaster misalignment between a user and *CA*.

Other objects, advantages and novel features, and further scope of applicability of the present invention will be set forth in part in the detailed description to follow, taken in conjunction with the accompanying drawings, and in part will become apparent to those skilled in the art upon examination of the following, or may be learned by practice of the invention. The objects and advantages of the invention may be realized and attained by means of the instrumentalities and combinations particularly pointed out in the appended claims.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated into and form a part of the specification, illustrate a preferred embodiment of the present invention and, together with the description, serve to explain the principles of the invention. The drawings are not to be construed as limiting the invention.

Fig. 1a is a diagrammatic overview of a central authority (*CA*) generating daemons to manage users' dynamic authentication keys (*DAK*s) in accordance with the present invention;

Fig. 1b is a diagrammatic overview of secure communication between users in accordance with the present invention;

Fig. 2 is a diagrammatic illustration of a user registration request to a *CA* in accordance with the present invention;

Fig. 3 is a diagrammatic overview of synchronization and authentication between users and a *CA*, and initial generation of the *DSK* by the *CA* in accordance with the present invention;

Fig. 4 is a diagrammatic illustration detailing the method of Fig. 3;

Fig. 5 is a diagrammatic illustration of synchronization of *DAK*s between a *CA* and a user in accordance with the present invention;

Fig. 6a is a diagrammatic illustration of the method whereby a *CA* authenticates a user in accordance with the present invention;

Fig. 6b is a diagrammatic illustration of the method whereby a user authenticates a *CA* in accordance with the present invention;

Fig. 7a is a diagrammatic illustration of the method whereby a *CA* freezes and resumes regeneration of users' *DAK*s upon occurrence of a *CA* shutdown event, in accordance with the present invention;

Fig. 7b is a diagrammatic illustration of the method whereby a user freezes and resumes regeneration of its *DAK* upon occurrence of a user shutdown event, in accordance with the present invention;

Fig. 8 is a diagrammatic illustration of secure communication establishment at the source user node in accordance with the present invention;

Fig. 9 is a diagrammatic illustration of secure communication establishment at the destination user node in accordance with the present invention;

Fig. 10 is a diagrammatic illustration of a user handshaking method with a *CA* in accordance with the present invention;

Fig. 11 is a diagrammatic illustration of the dynamic encryption and permutation method of the present invention;

Fig. 12 is a diagrammatic illustration of the dynamic decryption and permutation method of the present invention;

Fig. 13a is a diagrammatic illustration of the *DSK* regeneration method of the present invention;

Fig. 13b is a diagrammatic illustration of the *DSK* regeneration method of Fig. 13a demonstrating the utilization of previously encrypted data records from different time interval blocks in the regeneration process;

Fig. 14 is a diagrammatic illustration of the *DAK* regeneration method of the present invention;

Fig. 15a is a diagrammatic illustration of the formation of the auxiliary static key using the initial *DAK* in accordance with the present invention; and

Fig. 15b is a diagrammatic illustration of the formation of the auxiliary static key using the *DAK* and previous *DSK* in accordance with the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENTS (BEST MODES FOR CARRYING OUT THE INVENTION)

The present invention is a dynamic symmetric key encryption method and system for network security. The invention is implemented between end-users (*U*) and central authentication authorities (*CA*s) for user authentication purposes, and between end-users for secure exchange of digital data. The users and *CA* reside at their respective nodes, or programmable apparatuses, such as at one or more computers, and are in electrical communication, such as via a computer network. Communicated data flows over a data stream between the user and *CA* programmable apparatuses. Computer-readable memory provides storage for the data, dynamically changing keys, and other variables, as needed to allow for the regeneration of subsequent dynamic keys, and to carry out other necessary processes within the computer. Means are provided on the programmable apparatuses for performing all of the various methods involved in the dynamic

encryption method. Such means include primarily computer-readable means, such as software, and the necessary related hardware.

The encryption method of the present invention distributes the complexity of the encryption algorithm over the dynamics of the data exchange, involving previously encrypted data as well as the previous key in the process of regenerating a new "dynamic" encryption key. Thus, there is a newly generated key for the encryption of every data record, yielding zero entropy between the cipher and the plain data. There are no static keys, public or private, that are susceptible to a security breach. In order to guarantee security, the encryption method of the present invention is preferably deployed in a system of registered end-users with a *CA*, whereby the *CA* maintains user connections' authentication and secures distribution of symmetric encryption session keys.

The invention employs two types of dynamic keys, namely dynamic authentication keys (*DAK*s) and dynamic session keys (*DSK*s). The former is used to mutually authenticate users and *CA*s; it is continuously regenerated throughout the existence of the user and the *CA*. The latter exists when the need arises for a secure data exchange session between users; its regeneration is maintained only through the life cycle of such a session. The placement of the initial *DSK* at users' nodes is securely carried out by the *CA*, and encrypted using the users' *DAKs*. The *CA* maintains an array of *DAKs*, one per user.

The invention further employs an auxiliary static key *K*, which is formed based on the *DSK* and the *DAK*, given that the user has established a session; otherwise, it is formed from the initial *DAK* only. This static key is continuously involved in the regeneration of the *DAK*. The auxiliary static key adds another dimension of security to the process against insider attacks, as it involves more dynamics to the regeneration of the *DAK* by allowing the contribution of the *DSK* to the process, every new communication session between users. The static nature of *K* is not to be

exploited by the attacker since its exploitation does not lead to any important information; its relation to the *DSK* and the *DAK* is not reversible, i.e., *K* is manufactured from *DSK* and *DAK*, yet neither *DSK* nor *DAK* can be obtained from *K*.

The major role of the *CA* is to maintain the registered users' *DAK* generation and the secure delivery of symmetric session keys (*DSKs*) between the users. The *CA* also authenticates the source user to the destination user and vice versa, while authenticating itself to both users. Unless a user is pre-registered with the *CA*, it is nearly impossible for an intruder to have the same synchronized dynamic key with the *CA*, as a registered user. The only explicit user identity verification is carried out once, at the time the user first registers with the *CA*. Any consequent authentication is implicitly processed, without the need to exchange any plain keys, which would require a secure channel. The authentication method involves the three parties to any connection: source user, destination user, and *CA*, authenticating each other via their corresponding synchronized dynamic key.

Referring to Fig. 1a, a diagrammatic overview of a *CA* generating daemons to manage users' dynamic authentication keys (*DAKs*) in accordance with the present invention is shown. Registration of trusted users, users that have for example, provided a valid certificate of authentication or who are referred by a previously-registered third party user, occurs over a secure channel, for example, using RSA encryption, where the *CA* provides the user with an initial *DAK*. Referring to Fig. 1b, a diagrammatic overview of user communication encrypted with a dynamic session key (*DSK*) initially generated and sent by a *CA* is shown. Upon any user's connection request, a communication child process is forked at the *CA*, as well as at the user node. Each child process runs on the behalf of its parent during the whole communication session in order to avoid disturbance of the *DAK* generation process in the event of synchronization or authentication failure, for example, due to a false connection request, hardware failure, etc.

The *CA* is responsible for secure user authentication and generation of the initial trusted user dynamic authentication key, *DAK*. Every user must register with the *CA* in order to obtain its initial *DAK*, which is exchanged via a secure channel, e.g., using RSA. Upon the initiation of any new user's secure data exchange session, both end-users and the *CA* freeze their *DAK* generation and synchronize by establishing the same dynamic change. After the user and the *CA* are synchronized (their *DAK*s are aligned), they both yield the same randomly generated *DAK* at both nodes. This *DAK* is used in the authentication method as the encryption and decryption key, because it is maintained only by the *CA* and the user.

Referring to Fig. 2, a diagram further illustrates the user registration method with a *CA*. The user starts the registration process by sending a request **10**, effectively requesting the generation of an initial value of its *DAK*, to the *CA* including authenticated documentation of its identity and purpose of registration, i.e. revealing it is a trusted user. Upon approval of the user's credentials by the *CA*, the *CA* starts a daemon related to the requesting user, and randomly selects an initial *DAK*, sending a copy to the user via a secure channel **12**, for example, using the well-known RSA technique where the *CA* uses the user's public key to encrypt the newly generated *DAK*. Then the *CA* starts a daemon that permanently regenerates the *DAK*. Upon the reception of the initial *DAK*, the user starts a daemon in order to permanently regenerate the *DAK*. From that point forward, the user and *CA* randomly regenerate the next *DAK* every δt period, **14**, **15**, based on the previous generated *DAK* and the auxiliary static key *K*. The user and *CA* also maintain a number-regeneration-counter (*NRC*). This method of regenerating new *DAK*s is depicted in greater detail in Figs. 14 and 15.

Attention is now turned to Fig. 14. The diagram of Fig. 14 illustrates the continuous *DAK* regeneration method, where *DAK*[j] represents the j^{th} byte of the dynamic authentication key, *K*[j] represents the j^{th} byte of the auxiliary static key,

ExpK[*h*] represents the h^{th} byte of an "expanded key", $1 \le j \le m$, $1 \le h \le 2m$. Each unit *R* represents the random selection of one byte among the *ExpK*'s 2*m* bytes. An expanded key, *ExpK* **186** of twice the size of the *DAK* **182** is generated as follows. Each of the *DAK*, *K*, and *ExpK* are divided into (*m*/2) regions, indexed from 1 to (*m*/2). Each region of *DAK*, **182** and D*SK*, **184** is of length 2 bytes, and each region of *ExpK*, **186** is 4 bytes. The indices of the four consecutive bytes of any region *r* in the *ExpK* are indexed with the values: 4*r*-3, 4*r*-2, 4*r*-1, and 4*r*. The indices of the two consecutive bytes of any region *r*, in the *DAK* and *K*, are indexed with the values: 2*r*-1 and 2*r*. The four bytes of region *r* in the *ExpK* are filled from *DAK* and *K* as follows:

ExpK[4*r*-3] ← *DAK*[2*r*-1]

ExpK[4*r*-2] ← *DAK*[2r]

ExpK[4*r*-1] ← (*DAK*[2*r*-1]) XOR (*DAK*[2*r*])

ExpK[4*r*] ← (*DAK*[2*r*-1]) XOR (*DAK*[2*r*]) XOR (*K* [2*r*-1]) XOR (*K* [2*r*])

It will be understood by those of skill in the art that, *ExpK* can alternatively be comprised of any number of bytes, e.g., 2*m*, 3*m*, 8*m*, etc., and the invention is not limited to any particular size for *ExpK*. If *ExpK* were of a different, greater number of bytes than 2*m*, then the logic operation would be altered as needed to fill the bytes of *ExpK*. It will also be understood that the byte positions chosen to be XORed together to fill the positions of *ExpK* could similarly be altered, and still remain within the inventive scope of the dynamic encryption and authentication method. Similarly, alternative logic operations could be substituted for the XOR operation.

Once *ExpK* is created, a random selection of *m* bytes from the 2*m* bytes of *ExpK* is taken **190** based on the first byte of the *DAK* (*DAK*[1]) as the random function seed **188**. Alternatively, a randomly selected byte of the *DAK* can be used as the random function seed. This function generates a sequence of *m* random numbers, in

the range between 1 and $2m$, each of which represents the index of the *ExpK* byte to be placed as the next byte of the regenerated *DAK* **192**. The operation depicted in Fig. 14 is performed at both the user and the *CA* nodes. In order to maintain synchronization control, a number-regeneration-count for the dynamic authentication key (*DAK_NRC*) **194** is maintained and incremented after each *DAK* regeneration. This method can be performed periodically or aperiodically (with mutual consent of *CA* and the user) according to the implementation of the invention.

Continuing on to Figs. 15a and 15b, a diagram illustrates the method of forming the auxiliary static key *K*. Fig. 15b illustrates the formation of *K* using the *CA*-user aligned *DAK*, where *DAK*[*j*] represents the j^{th} byte of the dynamic authentication key, *DSK*[*j*] represents the j^{th} byte of the last dynamic session key, *K*[*j*] represents the j^{th} byte of the auxiliary static key *K*, and *ExpK*[*h*] represents the h^{th} byte of an "expanded key", $1 \le j \le m$, $1 \le h \le 2m$. Each unit *R* represents the random selection of one byte among the *ExpK*'s $2m$ bytes. Each static key *K* is created using the *CA*-user aligned *DAK*, **208** and the last user session *DSK*, **210**. First, an expanded key (*ExpK*), **212**, of twice the size of *K* is generated. Each of the *DAK* and the *DSK* is divided into (m/2) regions, indexed from 1 to (m/2). Each region of the *DAK* and the *DSK* is of length 2 bytes, and each region of *ExpK* is 4 bytes. The indexes of the four consecutive bytes of any region *r*, in the *ExpK*, are indexed with the values: $4r$-3, $4r$-2, $4r$-1, and $4r$. The indexes of the two consecutive bytes of any region *r*, in the *DAK* and the *DSK*, are indexed with the values: $2r$-1 and $2r$. The four bytes of region *r* in the *ExpK* are filled from *DAK* and *DSK* as follows:

$ExpK[4r\text{-}3] \leftarrow DAK[2r\text{-}1]$

$ExpK[4r\text{-}2] \leftarrow DAK[2r]$

$ExpK[4r\text{-}1] \leftarrow (DAK[2r\text{-}1])\ \text{XOR}\ (DAK[2r])$

$ExpK[4r] \leftarrow (DAK[2r\text{-}1])\ \text{XOR}\ (DAK[2r])\ \text{XOR}\ (DSK[2r\text{-}1])\ \text{XOR}\ (DSK[2r])$

Then, a random selection of *m* bytes from the 2*m* bytes of *ExpK* is performed, **216**, based on the *DAK*[1] byte as the random function seed, **214**. Alternatively, any randomly selected byte can serve as the random function seed. This function generates a sequence of *m* random numbers, in the range between 1 and 2*m*, each of which represents the index of the byte to be placed as the next byte of *K*, **218**. This operation is performed at both the source and destination user nodes. This method is used to form *K* when a user communication session is taking place and *DSK*s are being regenerated.

Fig. 15a illustrates the formation of *K*, **206** using the two copies of the initial *CA*-user aligned *DAK*, **196**, **198**. This method is used to form *K* when a communication session is not taking place, i.e., the user *DSK* does not exist. Thus, the mechanism of Fig. 15a follows the same pattern as Fig. 15b, except replacing the non-existing *DSK* by another copy of the initial *DAK*.

Returning to Fig. 3, a diagrammatic overview of synchronization and authentication between users and a *CA*, and initial generation of the *DSK* by the *CA* is shown. *DAK* regeneration starts at both the user and the *CA* nodes, upon user registration, where each generates the same sequence of random *DAK*s, with the initial *DAK* as a seed, even after the user-*CA* connection is terminated. Thus, key exchange between users, as well as between users and the *CA*, is minimized to maintain a high level of security. Users and the *CA* instead remain synchronized at all times with respect to *DAK* regeneration. When the user-*CA* connection is terminated after initial request for user registration with the *CA*, synchronized *DAK* regeneration continues off-line. Permanent regeneration of the *DAK* is maintained via a daemon running at each registered user, and a corresponding daemon running at the *CA*.

With continuing reference to the center portion of the diagram of Fig. 3, in order to establish a connection between a source user U_s and a destination user U_d, U_s's *DAK* regeneration daemon forks a child communication process U_s_*COM*, which freezes its version of the *DAK* and its *NRC*, and sends a connection request to the *CA* including synchronization information. Upon the reception of such request, the *CA*'s communication server forks a communication child process *CA_COM*, which will notify U_d of the connection with U_s. Upon the acceptance by U_d to the connection, U_d's *DAK* regeneration daemon forks a child communication process U_d_*COM*, which freezes its version of the *DAK* and its *NRC*, and sends synchronization information back to the *CA_COM*. Then, the *CA_COM* snapshots both users' *DAK*s/*NRC*s from their corresponding *CA*'s *DAK* daemons, and starts the synchronization and authentication processes with both users' communication child processes.

Upon successful mutual synchronization and authentication involving the three session parties, the *CA_COM* generates a random dynamic session key (*DSK)* and encrypts it using the aligned *DAK* of each user, and sends it to both users. After receiving the encrypted *DSK*, each of the two users' child communication process decrypts it using its respective aligned *DAK*, and starts a secure communication session with the other child communication process, via the decrypted *DSK*.

In case of failure, the child processes are terminated without interruption to the parent processes and/or daemons. This feature provides protection against a "synchronization disturbance" attack. In such an attack, an intruder imitating a registered user or a *CA* might force the three *DAK* number-regeneration-counters (*NRC*s) to freeze, which will create a chaotic state, but only in the child processes. The counters are continuously counting, i.e., *DAK*s are continuously regenerated, in the daemons without stopping, except when rebooting. The child processes snapshot, or "freeze", the *DAK/NRC* for use in the synchronization, authentication, and secure *DSK* exchange. Thus, the continuously running *DAK* daemons are

unaffected in the parent processes in the event of a failure to establish a connection, or in the event of a synchronization disturbance.

However, in the event of successful synchronization and authentication, the child processes at the users' nodes return the aligned *DAK* and the newly generated *DSK* to their respective parent daemons in order to initialize a new state for *DAK* regeneration, forcing the daemon to discard the current value of *DAK* and *DSK* (if exists), and consider the newly returned versions in the *DAK* regeneration process. Also, the *CA_COM* return the two aligned *DAKs* and the newly generated *DSK* to their respective *DAK* daemons at the *CA* in order to initialize a new state for *DAK* regeneration, forcing both local users' daemons to discard their current *DAK* and *DSK* (if exists) values, and consider the newly returned versions in the *DAK* regeneration process. Then, the *CA_COM* terminates successfully, whereas the U_s*_COM* and U_d*_COM* start a secure communication.

Referring to Fig. 4, a diagrammatic overview of synchronization, authentication, and generation of *DSK* by a *CA* in response to a request from a source user (U_s) to communicate with a destination user (U_d) is provided. (See also Fig. 3.) Source user U_s requests a DSK generation from *CA* to communicate with destination user U_d, and sends its frozen *DAK_NRC* along with the request, **16**. The *CA* forks a *CA_COM*, which snapshots the two users *DAKs*, namely *CA_DAK*[U_s] and *CA_DAK*[U_d], and requests U_d to send its *DAK_NRC*, **18**. This request notifies U_d that U_s is trying to establish a secure communication with it. Once the *CA* has received the U_d *DAK_NRC*, **20**, the synchronization process is initiated, **22**.

Synchronization ensures that the *CA* has *DAK* values identical to the users' *DAK* values, despite message propagation delay. The *CA* ensures that its locally snapshot *DAK* for U_s and the corresponding frozen *DAK* at U_s's node are aligned, and also ensures that its locally snapshot *DAK* for U_d and the corresponding frozen *DAK* at U_d's node, are aligned. To compensate for propagation delay and align the

corresponding *CA*'s *DAK* with each of the parties' *DAK*s, the difference (*x)* in the number of key regeneration counts (*NRCs*), between CA and each user, will be considered by the lagging party, which will regenerate its *DAK* an extra *x* times (See Fig. 5.)

If alignment is not achieved with both users, **26**, *CA* ignores the synchronization effects from the non-synchronized user(s), sends an abort message to both users before killing its communication child process, and cancels the communication due to lack of synchronization.

In the event of successful synchronization with both users, **24**, the *CA* launches an authentication method, **28**, to certify their identity. (Figs. 6a and 6b.) If successful authentication of both users is not achieved, **32**, *CA* ignores any synchronization effects of the non-authenticated user(s), sends an abort message to both users before killing its communication child process, and cancels the communication due to lack of authentication. If both users are fully authenticated and synchronized with *CA*, **30**, the *CA* randomly generates an initial *DSK* and sends it to both users, encrypted with each user's corresponding aligned *DAK*, to begin data exchange, **34**. This encryption process, **34**, is identical to the process described in Fig. 11, except that DSK_i is replaced by *DAK*, and D_i is replaced by the initial *DSK*. After the entire process is completed, successful or unsuccessful, the *CA_COM* terminates and returns its status to the parent process.

Referring to Fig. 5, a diagram illustrates synchronization of *DAK*s between a *CA* and a user, based on the number-regeneration-count for the *DAK* at each node. Initially, the number-regeneration-count of the *DAK* for user (*U*) at the *CA*, (*CA_DAK_NRC*[*U*]), is compared to the number-regeneration-count of the *DAK* at the user's node (*U_DAK_NRC*), **36**. If the two *NRC*s are equal, then the *CA* and user are synchronized. If the comparison of the *NRC*s is outside of a predetermined acceptable range, a "failure-to-synchronize" message is reported, **40**. This occurs

when |*CA_ DAK_NRC* [*U*] - *U_DAK_NRC*|, **38**, is larger than a predetermined value, and the communication is terminated.

If the comparison of the *NRC*s is within the predetermined acceptable range, then the lagging party performs a number of *DAK* regenerations equal to the calculated difference in order to synchronize (i.e., align the *DAK*s) with the other party. For example, if the *CA NRC* lags behind that of the user, **42**, then the *CA* performs (*U_DAK_NRC* - *CA_ DAK_NRC* [*U*]) regenerations of its *DAK* in order to align with that of *U*, **44**. If the user *NRC* lags behind that of the *CA*, the *CA* sends a "synchronize" message, including the calculated *NRC* difference, **46**, so that the user can perform the appropriate number of regenerations to synchronize with the *CA*. Once the user performs the regenerations, it signifies that it has done so to the *CA*, **48**.

Once the parties are synchronized, mutual authentication of *DAK*s is performed to ensure the parties indeed share the same *DAK*. Figs. 6a and 6b illustrate the mutual authentication method. Fig. 6a illustrates authentication of a user by a *CA*, and Fig. 6b illustrates authentication of a *CA* by a user. The process begins by *CA* generating a random number, or nonce, *N*. *CA* sends *N* and *E*(*N*) to the user, where *E*(*N*) is the encrypted version of *N* using the *CA_DAK*[*U*], the shared and aligned *DAK*, as well as an "authenticate" message, **50**. The user decrypts *E*(*N*) using its frozen *DAK*, which should be identical to *CA_DAK*[*U*], that of the *CA*, **64** and verifies that *D*(*E*(*N*))=*N*, **66**. The user thus authenticates the *CA*, **68**. If *D*(*E*(*N*)) does not equal *N*, then the user failed to authenticate the *CA* and the connection is aborted, **70**.

When the user has successfully authenticated the *CA*, **68**, the user encrypts N^2 with its *DAK*, ($E(N^2)$), and sends $E(N^2)$ back to the *CA*, as part of the authentication acknowledgment to complete the authentication process, **68**. Upon receiving the authentication acknowledgment back from the user, **52**, *CA* decrypts the received

ciphered number, again using the aligned *CA_DAK* [*U*], **54** and compares the decrypted value with the value of N^2, **56**. If they are not equal, the *CA* reports a failure to authenticate the user, **60**, and aborts the establishment of a connection. If they are equal, the user is successfully authenticated by the *CA*, i.e., has been registered with the *CA*, **58**. CA performs the same mutual authentication method with the second user before user-to-user communication takes place. When all parties have been mutually authenticated, *CA* proceeds to *DSK* generation, **62**, the last step of Fig. 4.

As systems are prone to unpredicted shutdown events, such as power loss, the dynamic encryption method and system automatically freezes and resumes key regeneration upon occurrence of such events. Referring to Fig. 7a, a diagram illustrating freezing and resuming regeneration of *DAK*s when a *CA* experiences a shutdown event, is shown. Fig. 7b shows freezing and resuming regeneration of *DAK*s when a user experiences a shutdown event. Referring to Fig. 7a, a *CA* is shown to experience a shutdown event, **72**. The *CA* immediately sends a "freeze-*DAK*-regenerating" message to all previously registered users, **74**, as part of its shutdown handler routine. Meanwhile, the *CA* saves all users' *DAK*s into a temporary file, **76**. After shutting down, **78**, for a time period τ, the *CA* reboots and reloads all the previously saved *DAK*s. The *CA* then requests the *DAK_NRC* from all users to ensure validity of the current *DAK*s, **80**. Then, the *CA* starts the synchronization process, **82** as described with respect to Fig. 5. The CA then initiates the mutual authentication process with all successfully synchronized users, **84**. (Figs. 6a and 6b.) Finally, the *CA* sends a "resume-*DAK*-regenerating" message to its registered and successfully synchronized and authenticated users, in order to resume realigned regeneration of the *DAK*s, **85**. A similar method is performed by the user in the event of a user's system shutdown, as depicted in Fig. 7b.

After a user is registered with a *CA*, the user may request a secure communication with another user. Referring to Fig. 8, a diagram illustrates secure communication establishment at the source user (U_s) side. The source user first determines whether it is registered to the *CA*, **86**. If not, the user performs the registration procedure and receives its initial *DAK* via a common secure channel, **88**. (See also Fig. 2.) Then, the source user's *DAK* daemon forks a child communication process U_s_*COM*, **90**, which freezes its *DAK* generation and runs on behalf of the user until the end of the users' session communication. U_s_*COM* requests a secure connection establishment with the destination user (U_d) from the *CA*, and sends its frozen *DAK_NRC* for synchronization purposes. In the event of successful handshaking with the *CA*, **92** (Fig. 10), the source user receives an initial dynamic session key *DSK*, from the CA, **94**.

The U_s–U_d data exchange session uses the shared symmetric *DSK* sent by the *CA* to both users. However, to increase the level of dynamic encryption to *n* crypto parallel streams, a set of *n* updated *DSK*s is derived randomly from the initial *DSK*, used as a seed, sent by *CA*. The source user message is stepped through *n* records, or a "block", at a time, and the first *n DSK*s generated are used to encrypt the first block of *n* records. After that, *n DSK*s are regenerated for each consecutive block as a function of previously generated *DSK*s and previously encrypted data records. (See Figs. 13a and 13b.)

This process is depicted in Fig. 8 after the source user has had successful handshaking, **94**. The source user first generates randomly *n* different *DSK*s (DSK_i, where $1 \le i \le n$), of the same size as the initial *DSK*, **96**. Then, *n* data records of the same size as the size of *DSK* ($Record_i$, where $1 \le i \le n$) are extracted from the input data, **98**, and encrypted **100**. Next, for every $Record_i$ a new DSK_i is regenerated, **102**, which is depicted in greater detail in Figs. 13a and 13b below on a byte-by-byte

basis. Finally, the n ciphered records are transmitted to U_d, **104**. The encryption method described at **100** of Fig. 8 is illustrated in greater detail in Fig. 11.

Turning to Fig. 11, a diagram illustrates the encryption method described at **100** of Fig. 8. In Fig. 11: D_i[j] is the j^{th} byte of the i^{th} data record, DSK_i[j] is the j^{th} byte of the corresponding i^{th} DSK, C_i^{tmp}[j] is the j^{th} byte of the i^{th} "temporary cipher record," C_i[j] is the j^{th} byte of the produced i^{th} cipher record, and PT_i[j] is the j^{th} byte of the i^{th} permutation table, $1 \le i \le n$ and $1 \le j \le m$, for n records, each of which is of m bytes.

The data record D_i and the DSK_i are used to produce C_i^{tmp}, which is permuted based on PT_i to produce cipher record C_i, $1 \le i \le n$, producing n cipher records. The encryption method uses a simple XOR logic operation as an encryption function between a DSK_i and its corresponding data record D_i. The data byte D_i[j], **146**, is XORed with the corresponding key byte DSK_i[j], **148**, resulting in a temporary cipher byte C_i^{tmp}[j], **150**. To shuffle the C_i^{tmp} bytes around, a permutation table PT_i, **152** is generated as a copy of the corresponding DSK_i. The PT_i bytes are scanned from index 1 to index m, using each entry's index and its associated entry value, as indices of two bytes in the temporary cipher record to be swapped. The permutation is performed as follows: permute C_i^{tmp}[j] with C_i^{tmp}[PT_i[j]], for $1 \le j \le m$, which results in the final cipher record C_i, **156**. After n data records are encrypted with n DSKs as aforementioned, the final cipher block, made up of n cipher records (C_i, where $1 \le i \le n$), is available for transmission, and a set of n new DSKs is regenerated for encryption of the next data block, as discussed below with reference to Figs. 13a and 13b.

Returning to Fig. 8, the final cipher block is transmitted to U_d, **104** after having generated the n updated DSKs, **102**, for encrypting the next data block of n records. The method of data encryption and transmission is repeated, **106**, until the entire data volume has been encrypted and transmitted.

The decryption method is depicted in Fig. 9 at the destination side, U_d. Initially, U_d receives a communication request from the *CA*, **108**. Then its *DAK* daemon forks a child communication process U_d _*COM* in order to freeze the generation of its version of *DAK* and sends the *DAK_NRC* to the *CA* for synchronization purposes, **110**. U_d _*COM* will run on behalf of the user until the end of the users' session communication. In the event of successful handshaking with the *CA*, **112** (Fig. 10), the destination user receives an initial dynamic session key *DSK*, from the CA, **114**.

After successful handshaking, U_d uses the initial *DSK* to randomly generate *n* *DSK*s ($DSK^t{}_i$; $1 \le i \le n$) of the same size as the initial *DSK* (used as a seed), **116**. Beginning with the same initial *DSK* at both U_s and U_d sites, U_d randomly derives the same set of *n* regenerated *DSK*s as the source user U_s, and parallels the encryption method described at the source user side in reverse, to decrypt the transmitted data. U_d receives a block of *n* cipher records ($Cipher_i$; $1 \le i \le n$) **118**. All cipher records are decrypted using each record's corresponding *DSK*, **120**. Concatenation of the decrypted records provides the original message data to the destination user, **122**. U_d then regenerates new *DSK*s from the recovered block of data records and the current *DSK*s to be used for decryption of the next block of cipher records, **124**, as illustrated in Figs. 13a and 13b. The process is repeated, **126**, until all the transmitted cipher data has been decrypted.

Referring to Fig. 12, a diagram illustrates the decryption method at **120** of Fig. 9 in greater detail. To shuffle the *m* bytes of each of the received *n* cipher records, **158**, back to the correct order, a permutation table PT_i is generated for each cipher record C_i, **160**. The permutation table is generated, as at the source side, as a copy of the corresponding *DSK*. Permutation is performed on the received cipher record bytes using each of the table entry's index and its associated entry value, as indices of the two bytes in the cipher record to be swapped, but in bottom-up order, **162**: permute $C_i[j]$ with $C_i[PT_i[j]]$, for j from m down to 1. The result is a temporary cipher record

C_i^{tmp}, **164**. The decryption method continues by performing the XOR operation on C_i^{tmp}[j], **164** and the corresponding DSK_i[j], for $1 \leq j \leq m$,**166**. This results in the original data record D_i, for $1 \leq i \leq n$, **168**. The process continues as shown in Fig. 9, at **122**.

As discussed in Figs. 8 and 9, upon request for a secure communication, both the source user and destination user must accomplish a successful handshaking process with the *CA*. Fig. 10 illustrates a user handshaking method with the *CA*. Upon receiving a message from the *CA*, **130**, the user responds accordingly, **132**. If an "abort" message is received, then the user terminates the communication child process in charge of the connection, **144**. If authentication is requested, **138**, the user mutually authenticates with the *CA* as described in Fig 6(b). If a "synchronize" message is received including a number (x), the user performs x regenerations of its *DAK*, in order to be synchronized with the *CA*, i.e., aligning to the same *DAK* **134**, as described in Fig. 5. An acknowledgment message is then sent back to the *CA*, **136**.

After either synchronization or authentication, the user waits for a "session key" message including an encrypted *DSK*, (*E*(*DSK*)). The encrypted *DSK* is decrypted, **140**. Then, the communication child process returns the aligned *DAK* and the new *DSK* to its parent daemon in order to initialize a new state for the *DAK* regeneration state, **141**, and the communication between the source and destination is securely established, **142**. The process continues at **96** of Fig. 8 and **116** of Fig. 9.

Attention is now turned to Figs. 13a and 13b, where regeneration of *n DSK*s is shown. Fig. 13a is a diagram illustrating the dynamic session key (*DSK*) regeneration method of Fig. 8, **102**, and Fig. 9, **124**. Fig. 13b is a diagram illustrating *n* "tracks" or streams of parallel *DSK* regeneration for a plurality of data record blocks. Each block of data records includes those data records encrypted within a predefined time increment (T). Fig. 13b illustrates blocks of data records from the range of time increments of $T = 1$ through $T = t + 1$. A "track" is defined as those data records in

alignment within their respective data blocks, hence a track consists of a vertical column of data records as depicted in Fig. 13b.

In Fig. 13a $DSK^t_i[j]$ represents the j^{th} byte of the last i^{th} dynamic session key at time t, $D^{t\_old}_i[j]$ represents the j^{th} byte of the last i^{th} data record at time "t_old", and $ExpK^t_i[h]$ represents the h^{th} byte of the i^{th} "expanded key" at time t, $1 \le i \le n$ and $1 \le j \le m$, $1 \le h \le 2m$. A data record $D^{t\_old}_i$ is selected randomly from the previously encrypted data records $\{D^1_i, D^2_i, \ldots, D^{t-1}_i\}$ in the i^{th} track. This is accomplished by randomly selecting the index t_old from the range of [1, t-1] using a predetermined byte of the current key $DSK^t_i[j]$ as the seed of random generation, **176**. For example, the first byte of the current key, $DSK^t_i[1]$, may be used as the seed of random generation. However, when generating DSK^2_i, the only available data record is D^1_i, therefore, t_old = 1. Once the index t_old is selected from the range [1, t-1], the corresponding previously encrypted data record $D^{t\_old}_i$ **172** is XORed with the *DSK* as shown.

Next, an expanded key ($ExpK^t_i$), **174**, of twice the size of DSK^t_i is generated. Each of DSK^t_i, $D^{t\_old}_i$, and $ExpK^t_i$ is divided into (m/2) regions, indexed from 1 to (m/2). Each region of DSK^t_i and $D^{t\_old}_i$ is of length 2 bytes, and each region of $ExpK^t_i$ is 4 bytes. The indexes of the four consecutive bytes of any region r, in the $ExpK^t_i$, are indexed with the values: 4r-3, 4r-2, 4r-1, and 4r. The indexes of the two consecutive bytes of any region r, in the DSK^t_i and $D^{t\_old}_i$, are indexed with the values: 2r-1 and 2r. Filling the four bytes of region r in the $ExpK^t_i$, from DSK^t_i and $D^{t\_old}_i$ is defined as follows:

$$ExpK^t_i[4r\text{-}3] \leftarrow DSK^t_i[2r\text{-}1]$$

$$ExpK^t_i[4r\text{-}2] \leftarrow DSK^t_i[2r]$$

$$ExpK^t_i[4r\text{-}1] \leftarrow (DSK^t_i[2r\text{-}1]) \text{ XOR } (DSK^t_i[2r])$$

$$ExpK^t_i[4r] \leftarrow (DSK^t_i[2r\text{-}1]) \text{ XOR } (DSK^t_i[2r]) \text{ XOR } (D^{t\_old}_i[2r\text{-}1]) \text{ XOR } (D^{t\_old}_i[2r])$$

Then, a random selection of m bytes from the $2m$ bytes of $ExpK^t_i$ is performed, **178**, using the first byte $DSK^t_i[1]$ as the random function seed, **176**. Alternatively, any byte $DSK^t_i[j]$, can serve as the random function seed. This function generates a sequence of m random numbers, in the range between 1 and $2m$, each of which represents the index of the byte to be placed as the next byte of the newly generated DSK^{t+1}_i, **180**. This operation is performed at both the source and destination user nodes. The method can be performed periodically or aperiodically (with mutual consent of source and destination), depending on the implementation of the invention.

Referring to Fig. 13b, the initial *DSK* is first received from the *CA* in order to start a secure source-destination communication with n tracks of parallel encryption, each track having its own stream of *DSK* keys. The initial DSK_{CA} received from the *CA* is used to randomly generate the initial n *DSK*s, one for each of the n tracks of data records, $(D^1_1, D^1_2, \ldots, D^1_n)$, where D^t_i represents the data record of the i^{th} track at time t, and DSK^t_i represents the *DSK* of the i^{th} track at time t. (See also Fig. 8, **96** and Fig. 9, **116**.) At time t+1, the DSK^{t+1}_i is regenerated based on DSK^t_i and a randomly selected data record from the previously encrypted data record set $\{D^1_i, D^2_i, \ldots, D^{t-1}_i\}$ in the i^{th} track. As stated, the data record is randomly selected by randomly selecting the index t_old from the range of [1, t-1] using a predetermined byte of the current key $DSK^t_i[j]$ as the seed of random generation. In Fig. 13b, "R" represents a random selection operation while "RF" represents the regeneration function of Fig. 13a.

A randomly selected encrypted data record from the data record set $\{D^1_i, D^2_i, \ldots, D^{t-1}_i\}$ is used in the regeneration of DSK^{t+1}_i to render the latter obsolete in the event that a *DSK* is compromised by unauthorized intrusion. In the unlikely event that DSK^{t+1}_i, in the i^{th} track, is obtained by an intruder, then the intruder can only obtain

the data record D_i^{t+1} because the intruder has the cipher C_i^{t+1}. However, the intruder cannot obtain more than that. The intruder must obtain the entire set of previously encrypted data records $\{D_i^1, D_i^2, \ldots, D_i^t\}$ in order to generate DSK_i^{t+2}, or any successor *DSK*. Hence, breaking one *DSK* will aid only in decrypting its corresponding data record and nothing more, past or future ciphered data.

It will be understood by those of skill in the art that *ExpK* can alternatively be comprised of any number of bytes, and the invention is not limited to any particular size for *ExpK*. It will also be understood that the byte positions chosen to be XORed together to fill the positions of the *ExpK* of Figs. 13 and 14 could similarly be altered, and still remain within the inventive scope of the dynamic encryption and authentication method. It will also be understood that the size of the regions of the keys used to regenerate new *DSKs* and *DAK*s, need not be 2 or 4 bytes, but could be of any size within the size of the respective keys. Similarly, alternative logic operations could be substituted for the XOR operation.

Although the invention has been described in detail with reference to this preferred embodiment, other embodiments can achieve the same results. Variations and modifications of the present invention will be obvious to those skilled in the art and it is intended to cover in the appended claims all such modifications and equivalents. The entire disclosures of all references, applications, patents, and publications cited above are hereby incorporated by reference.